ELSEVIER

Trawling the Web for emerging cyber-communities

Ravi Kumar [1], Prabhakar Raghavan [*,1], Sridhar Rajagopalan [1], Andrew Tomkins [1]

IBM Almaden Research Center K53, 650 Harry Road, San Jose, CA 95120, USA

Abstract

The Web harbors a large number of communities — groups of content-creators sharing a common interest — each of which manifests itself as a set of interlinked Web pages. Newgroups and commercial Web directories together contain of the order of 20,000 such communities; our particular interest here is on emerging communities — those that have little or no representation in such fora. The subject of this paper is the systematic enumeration of over 100,000 such emerging communities from a Web crawl: we call our process trawling. We motivate a graph-theoretic approach to locating such communities, and describe the algorithms, and the algorithmic engineering necessary to find structures that subscribe to this notion, the challenges in handling such a huge data set, and the results of our experiment.

Keywords: Web mining; Communities; Trawling; Link analysis

1. Overview

The Web has several thousand well-known, explicitly defined *communities* — groups of individuals who share a common interest, together with the Web pages most popular amongst them. Consider for instance, the community of Web users interested in Porsche Boxster cars. There are several explicitly gathered resource collections (e.g., Yahoo's Recreation: Automotive: Makes and Models: Porsche: Boxster) devoted to the Boxster. Most of these communities manifest themselves as newsgroups, Webrings, or as resource collections in directories such as Yahoo! and Infoseek, or as homesteads on Geocities. Other examples include popular topics such as 'Major League Baseball,' or the somewhat less visible community of 'Prepaid phone card collectors'. The explicit nature of these communities makes them easy to find — it is simply a matter of visiting the appropriate portal or newsgroup. We estimate that there are around twenty thousand such explicitly defined communities on the Web today.

The results in this paper suggest that the distributed, almost chaotic nature of content-creation on the Web has resulted in many more *implicitly defined communities*. Such implicitly defined communities often focus on a level of detail that is typically far too fine to attract the current interest (and resources) of large portals to develop long lists of resource pages for them. Viewed another way, the methods developed below identify Web communities at a far more nascent stage than do systematic and institutionalized ontological efforts. (The following preview of the kinds of communities we extract from the Web may underscore this point: the community of Turkish student organizations in the US, the community centered around oil spills off the coast

* Corresponding author.
[1] E-mail: {ravi,pragh,sridhar,tomkins}@almaden.ibm.com

of Japan, or the community of people interested in Japanese pop singer *Hekiru Shiina*.)

There are at least three reasons for systematically extracting such communities from the Web as they emerge. First, these communities provide valuable and possibly the most reliable, timely, and up-to-date information resources for a user interested in them. Second, they represent the sociology of the Web: studying them gives insights into the intellectual evolution of the Web. Finally, portals identifying and distinguishing between these communities can target advertising at a very precise level.

Since these implicit communities could outnumber the explicit ones by at least an order of magnitude, it appears unlikely that any explicitly defined manual effort can successfully identify and bring order to all of these implicit communities, especially since their number will continue to grow rapidly with the Web. Indeed, as we will argue below, such communities sometimes emerge in the Web even before the individual participants become aware of their existence. The question is the following: can we automatically recognize and identify communities that will clearly pass under the radar screen of any human ontological effort?

The subject of this paper is the explicit identification of (a large number of) these implicit communities through an analysis of the Web graph. We do not use search or resource-gathering algorithms that are asked to find pages on a specified topic. Instead, we scan through a Web crawl and identify *all* instances of graph structures that are indicative signatures of communities. This analysis entails a combination of algorithms and algorithmic engineering aimed at enumerating occurrences of these subgraph signatures on the Web. We call this process *trawling* the Web.

In the process of developing the algorithm and system for this process, many insights into the fine structure of the Web graph emerge; we report these as well.

1.1. Related prior work

Link analysis. A number of search engines and retrieval projects have used links to provide additional information regarding the quality and reliability of the search results. See for instance [4,5,8,9,14]. Link analysis has become popular as a search tool. However, our focus in this article is in using link analysis for a different purpose: to understand and mine community structure on the Web.

Information foraging. The information foraging paradigm was proposed in [19]. In their paper, the authors argue that Web pages fall into a number of types characterized by their role in helping an information forager find and satisfy his/her information need. The thesis is that recognizing and annotating pages with their type provides a significant 'value add' to the browsing or foraging experience. The categories they propose are much finer than the hub and authority view taken in [14], and in the Clever project. Their algorithms, however, appear unlikely to scale to sizes that are interesting to us.

The Web as a database. A view of the Web as a semi-structured database has been advanced by many authors. In particular, LORE (see [1]) and WebSQL (see [18]) use graph-theoretic and relational views of the Web. These views support a structured query interface to the Web (Lorel and WebSQL, respectively) which is evocative of and similar to SQL. An advantage of this approach is that many interesting queries, including methods such as HITS, can be expressed as simple expressions in the very powerful SQL syntax. The associated disadvantage is that the generality comes with a computational cost which is prohibitive in our context. Indeed, LORE and WebSQL are but two examples of projects in this space. Some other examples are W3QS [15], WebQuery [7], Weblog [17], and ParaSite [20].

Data mining. Traditional data mining research, see for instance [2], focuses largely on algorithms for inferring association rules and other statistical correlation measures in a given data set. Our notion of trawling differs from this literature in several ways.

(1) We are interested in finding structures that are relatively rare — the graph-theoretic signatures of communities that we seek will perhaps number only a handful — for any single community. By contrast, data mining tools look for patterns based on support and confidence.
(2) An exhaustive search of the solution space is infeasible (even with efficient methods such as *a priori* [2] or Query Flocks [21]); unlike market baskets, where there are at most about a mil-

lion distinct items, there are between two and three orders of magnitude more 'items', i.e., Web pages, in our case.

(3) The relationship we will exploit, namely co-citation, is effectively the join of the Web 'points to' relation and its transposed version, the Web 'pointed to by' relation. The size of this relation is potentially much larger than the original 'points to' relation. Thus, we need methods that work implicitly with the original 'points to' relation, without ever computing the co-citation relation explicitly.

Algorithmic and memory bottleneck issues in graph computations are addressed in [13]. The focus in this work is in developing a notion of *data streams*, a model of computation under which data in secondary memory can be streamed through main memory in a relatively static order. They show a relationship between these problems and the theoretical study of *communication complexity*. They use this relationship mainly to derive impossibility results. Gibson et al. [12] describe experiments on the Web in which they use spectral methods to extract information about 'communities' in the Web. They use the non-principal eigenvectors of matrices arising in Kleinberg's [14] HITS algorithm to define their communities. They give evidence that the non-principal eigenvectors of the co-citation matrix reveal interesting information about the fine structure of a Web community. While eigenvectors seem to provide useful information in the contexts of search and clustering in purely text corpora (see also [10]), they can be relatively computationally expensive on the scale of the Web. In addition they need not be complete, i.e., interesting structures could be left undiscovered. The second issue is not necessarily a show-stopper, as long as not too many communities are missed. The complementary issue, 'false positives', can be problematic. In contrast, our results below indicate that we almost never find 'coincidental' false positives. For a survey of database techniques on the Web and the relationships between them, see [11].

1.2. Strongly connected bipartite subgraphs and cores

Websites that should be part of the same community frequently do not reference one another. In well-known and recognized communities, this happens at times for competitive reasons and at others because the sites do not share a point of view. Companies competing in the same space, AT&T and Sprint for instance, do not point to each other. Instances of the second kind are pages on opposite sides of a thorny social issue such as gun control or abortion rights. In the case of emerging communities this happens for a third, more mundane, reason. Even pages that would otherwise point to one another frequently do not — simply because their creators are not aware of each others' presence.

Linkage between these related pages can nevertheless be established by a different phenomenon that occurs repeatedly: *co-citation*. For instance www.swim.org/church.html, www.kcm.co.kr/search/church/korea.html, and www.cyberkorean.com/church all contain numerous links to Korean churches (these pages were unearthed by our trawling). Co-citation is a concept which originated in the bibliometrics literature (see for instance, [22]). The main idea is that pages that are related are frequently referenced together. This assertion is even more true on the Web where linking is not only indicative of good academic discourse, but is an essential navigational element. In the AT&T/Sprint example, or in the case of IBM and Microsoft, the corporate home pages do not reference each other. On the other hand, these pages are very frequently 'co-cited.' It is our thesis that co-citation is not just a characteristic of well-developed and explicitly known communities but an early indicator of newly emerging communities. In other words, we can exploit co-citation in the Web graph to *extract all communities that have taken shape on the Web, even before the participants have realized that they have formed a community*.

There is another property that distinguishes references in the Web and is of interest to us here. Linkage on the Web represents an implicit endorsement of the document pointed to. While each link is not an entirely reliable value judgment, the sum collection of the links are a very reliable and accurate indicator of the quality of the page. Several systems — e.g., HITS, Google, and Clever — recognize and exploit this fact for Web search. Several major portals also apparently use linkage statics in their ranking functions because, unlike text-only ranking functions, linkage statistics are relatively harder to 'spam'.

Thus, we are left with the following mathematical version of the intuition we have developed above. *Web communities are characterized by dense directed bipartite subgraphs*. A bipartite graph is a graph whose node set can be partitioned into two sets, which we denote F and C. Every directed edge in the graph is directed from a node u in F to a node v in C. A bipartite graph is dense if many of the possible edges between F and C are present. Without mathematically pinning down the density, we proceed with the following hypothesis: the dense bipartite graphs that are signatures of Web communities contain at least one *core*, where a core is a complete bipartite subgraph with at least i nodes from F and at least j nodes from C (recall that a complete bipartite graph on node-sets F, C contains all possible edges between a vertex of F and a vertex of C). Note that we deviate from the traditional definition in that we allow edges within F or within C. For the present, we will leave the values i, j unspecified. Thus, the core is a small (i, j)-sized complete bipartite subgraph of the community. We will find a community by finding its core, and then use the core to find the rest of the community. The second step — finding the community from its core can be done, for instance, by using an algorithm derived from the Clever algorithm. We will not describe this latter step in detail here. We will instead focus the rest of this paper on the more novel aspect, namely algorithms for finding community cores efficiently from the Web graph, and the potential pitfalls in this process. Our algorithms make heavy use of many aspects of Web structure that emerged as we developed our algorithms and system; we will explain these as we proceed.

We first state here a fact about random bipartite graphs. We then derive from it a less precise hypothesis that, by our hypothesis, applies to Web communities (which cannot really be modeled as random graphs). We do this to focus on the qualitative aspects of this imprecise hypothesis rather than its specific quantitative parameterization.

Fact 1. Let B be a random bipartite graph with edges directed from a set L of nodes to a set R of nodes, with m random edges each placed between a vertex of L and a vertex of R uniformly at random. Then there exist i, j that are functions of $(|L|, |R|, m)$ such that with high probability, B contains i nodes from L and j nodes from R forming a complete bipartite subgraph.

The proof is a standard exercise in random graph theory and is omitted; the details spelling out the values of i and j, and 'high probability' can be derived easily. For instance, it is easy to show that if $|L| = |R| = 10$ and $m = 50$, then with probability more than *0.99* we will have i and j at least 5. Thus, even though we cannot argue about the distribution of links in a Web community, we proceed with the following hypothesis.

Hypothesis 1. *A random large enough and dense enough bipartite directed subgraph of the Web almost surely has a core.*

The most glaring imprecision in the hypothesis is what is meant by a dense enough and large enough random bipartite subgraph. Indeed, answering this question precisely will require developing a random graph model for the Web. The second source of imprecision is the phrase 'almost surely'. And finally we leave unspecified the appropriate values of i and j. We state this hypothesis largely to motivate finding large numbers of Web communities by enumerating all cores in the Web graph; we use our experiments below to validate this approach. Note that a community may have multiple cores, a fact that emerges in our experiments. Note also that the cores we seek are *directed* — there is a set of i pages all of which hyperlink to a set of j pages, while no assumption is made of links *out* of the latter set of j pages. Intuitively, the former are pages created by members of the community, focusing on what they believe are the most valuable pages for that community (of which the core contains j). For this reason we will refer to the i pages that contain the links as *fans*, and the j pages that are referenced as *centers* (as in community centers).

2. Web structure and modeling

In the course of designing an algorithm that trawls the Web graph for community cores, we discovered several interesting phenomena which also guided algorithmic choices. We detail some of these in this section. We begin with a description of our data source.

2.1. Data source and resources

Our data source is a copy of the Web from Alexa, a company that archives the state of the internet. The crawl is over a year and a half old, and consequently somewhat out of date. However, for our purposes, this is a very interesting data set because it allows us to discover communities emerging on the Web a year ago and then compare them against the state of the Web today (when many of these communities should be better developed). Indeed, one of the most interesting consequences of our work is the inherent resilience in the process: it is relatively easy to track down today's community from its *fossil* — the core we extract from the year-and-a-half-old copy of the Web — even though some of the pages in the core may no longer exist. We will revisit this theme later in the paper.

The raw Web crawl consisted of about 1 Terabyte of Web data on tape. The data was text-only HTML source and represented the content of over 200 million Web pages, with the average page containing about 5 Kbytes of text. Compared to the current size of the Web (see [3]) the volume of data is significantly smaller. This difference, however, should not be an impediment to implementing many of our methods on the entire Web, as we will point out below. All our experimentation was done on a single IBM PC with an Intel 300 MHz Pentium II processor, with 512 MB of memory, running Linux. The total running time of our experiment was under two weeks.

2.2. Fans

The first step of the algorithm is to scan through the data set and summarize the identities and content of all *potential fans*. For this purpose, we must pinpoint a notion of a potential fan. We may view *fans* as specialized *hubs* — extensive resource lists discovered by algorithms such as HITS and Clever. We looked at the outputs of over 500 good Clever search results that were created during a series of taxonomy-creation experiments. We analyzed the hubs in these results, looking for a syntactic, but principled, notion that characterizes good hubs. A reliable characteristic of good hubs was that most contained many *non-nepotistic* links to good resources. By non-nepotistic links, we mean links to pages on other sites. Thus, we chose the following syntactic definition for a potential fan page: a *potential fan page has links to at least 6 different Websites*. For the purposes of this definition, a Website is the first field of the URL. Of the 200 million original Web pages approximately 12%, i.e., about 24 million pages, were extracted from the tapes as potential fan pages. For each of these 24 million pages, we retained only the sequence of hyperlinks occurring in that page, discarding all other information. This is in keeping with our objective of using purely link information in this experiment. One could conjecture that content and text information could be further used to identify interesting communities, fearing that without content analysis many communities suggested by the graph structure would turn out to be coincidences. There are two reasons we have not done so in the current experiment.

(1) We want to study how far one could go with linkage information alone, since the efficiency improvements from retaining only the link information on pages is significant (and indeed makes an experiment like ours feasible on the scale of the entire Web).
(2) We began with the thesis that for moderate values of i and j (say, in the range 3 to 9), most of the cores identified by graph analysis alone would in fact correspond to real communities rather than coincidences; as our results below show, this belief is amply vindicated in that almost none of our cores is spurious.

Henceforth when we speak of our operations using fan pages, we will think of a fan page as a sequence of links devoid of all other content. At any point in our trawling process, the set of potential fans remaining in contention implies a set of potential centers remaining in contention: namely, those links listed in the current set of potential fans. Folding page-content into trawling is an interesting issue and one which is the subject of ongoing research at the Almaden Research Center.

2.3. Mirrors and shingles

Many existing communities are mirrored repeatedly, both in their fans and in their centers. To some extent this is inevitable and is a testament to the value of resource pages that bring together the pages in a community. There are surprisingly many mir-

rors of many Yahoo! pages, not all of which are owned by the many avatars of Yahoo. Another phenomenon, observed previously by [6], is that content on the Web is rather easy to copy and reproduce with minor variations. Many pages are reproduced in slightly modified forms (for instance, the claimed author is different from one copy to another). Broder et al. [6] propose a *shingling* method to identify and eliminate such duplicates, which we adopt as well, except that we only apply it to the sequence of links in a page (rather than to the entire page, as they do). The method constructs a number of local hashes of the Web page and compares smallest few hash values (called *shingles*) to detect duplication. In their paper, Broder et al. [6] argue that if the hash function and the number of shingles are chosen carefully, then with high probability, exact and almost-exact duplicates are detected. On the other hand, two very different pages are almost never accidentally accused of duplication.

In our case, however, even approximate mirroring that only preserves most of the links (say 90% of them) can be fatal. A page that is approximately mirrored say three times can produce a very large spurious core, e.g., a $(3, k)$ core where k can be large (here k would be the number of links preserved in all the mirrorings). Even if, say, one in a hundred of our potential fans were approximately duplicated in this fashion, we would have as many spurious communities as real ones.

Consequently, we choose a very aggressive mirror-elimination strategy. The number of *shingles* we maintain is rather small: only two per page. We also use a relatively small local window (five links) over which to compute the shingles. While these aggressive choices for the shingle algorithm parameters can and probably do result in a few distinct pages being misidentified as mirrors, it detects almost all mirrors and near mirrors reliably. Thus, we are able to effectively deal with the problem posed by near mirrors generating spurious communities.

Of the 24 million pages that were chosen as potential fans, mirror elimination based on shingling removed 60% of the pages (this refers to the number of potential fans; the number of potential centers remaining is roughly 30 times the number of potential fans, at all stages). A natural question would be whether the shingling strategy was overly aggressive. To convince ourselves of this we examined by hand a sample of the deleted pages and found that in almost every case, the deleted page was indeed a mirror page. Indeed we found many of the duplicates were due to idiosyncrasies in the crawl. For instances, URLs such as http://foo.com/x/y/and http://foo.com/x/y/../y/ (clearly the same) were treated as distinct. Another problem was the plethora of plagiarized Yahoo! pages. Many instances of Yahoo! pages had as many as 50 copies. Here is an instance of four pages in the crawl whose content is almost the same.

http://www.sacher.com/~schaller/shop/shop_mo.htm
http://www.assoft.co.at/~schaller/shop/shop_mo.htm
http://www.technosert.co.at
/~schaller/shop/shop_mo.htm
http://www.der-reitwagen.co.at
/~schaller/shop/shop_mo.htm

Even after mirror deletion, one cannot hope (especially given the constraints of memory hierarchies) to efficiently enumerate all cores using a classical relational approach (as, say, an SQL statement in a database, or its Web equivalent, WebSQL). If n were to denote the number of pages remaining, and i were the size of the core, this would lead to a running time that grew roughly as n^i, which is prohibitive for any i larger than 1. It is therefore essential to better exploit the detailed characteristics of the Web graph to prune the data down, eliminating those potential fans that can be proved not to belong to any community; we begin describing this process now. Indeed, inferring and understanding some of these characteristics is a fascinating study of the Web graph of interest in its own right.

2.4. In-degree distribution

Our first approach to trimming down the data resulted from an analysis of the in-degrees of Web pages. The distribution of page in-degrees has a remarkably simple law, as can be seen in Fig. 1.

This chart includes pages that have in-degree at most 410. For any integer k larger than 410, the chance that a page has in-degree k is less than 1 in a million. These unusually popular pages (such as www.yahoo.com) with many potential fans pointing to them have been excluded. The chart suggests a

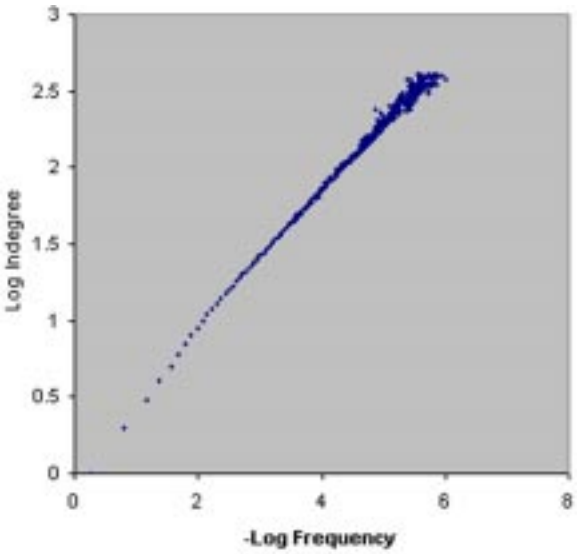

Fig. 1. In-degree distribution.

simple relation between in-degree values and their probability densities. Indeed, as can be seen from the remarkably linear log–log plot, the slope of the curve is close to $1/2$. This allows us to state the following empirical fact.

Empirical fact: *The probability that a page has in-degree i is roughly $1/i^2$.*

The precise exponent is slightly larger than 2. For our purposes, however, 2 is close enough. Also, by elementary probability, we see that the chance that a page has degree at least i is proportional to $1/i$.

2.5. Pruning centers by in-degree

As we have argued earlier, known and established communities typically contain relatively dense bipartite cores. (Indeed, this is one of the reasons that algorithms such as HITS and Clever work very well on broad topic queries, for which there is a significant Web presence in terms of number of pages that address the topic.)

However, large and dense bipartite graphs can and do contain many instances of the small cores that we are looking for. This creates an implementation problem: if an algorithm were to list all possible cores in the Web graph, then we could be stuck in a situation where most of the cores corresponded to very 'high-level' communities (e.g., Physics) and would leave us with a needle-in-the-haystack problem to find and distinguish those that were emerging or new. *Pruning by in-degree* is a simple method of addressing this potential problem. We delete all pages that are very highly referenced on the Web, such as the home pages of Web portals (e.g., Yahoo! or Altavista). These pages are presumably referenced for a variety of reasons not having to do with any single emerging community, and hence can safely be eliminated from further consideration. (On the other hand if we were to retain them, we raise the odds of discovering spurious 'communities' because pages on various subjects may contain links to these sites just for the creators' convenience.) We therefore eliminate (as potential centers) all pages that have an in-degree greater than a carefully chosen threshold k. The issue, then, is the particular choice of k.

We posit that pages listed in a Web directory such as Yahoo! are relatively uninteresting from our point of view. This is because these pages belong to communities that are already developed and explicitly known. We further note that directory services listing explicitly known communities, like Yahoo!, list about 10 million pages. Let us further approximate by 400 million the order of magnitude of the number of pages in the Web today. Thus, the chance that a page is already known to be part of an explicitly known community is about 1 in 40. From the empirical fact, such a node would have in-degree 40 or larger. The exact constants turn out not to be critical; the calculation above shows that the correct value is around 40. We conservatively set k to 50, and prune all pages that have in-degree 50 or larger from further consideration as centers.

3. Trawling algorithms and system

Thus far we have described several preliminary processing steps on our data, along with some interesting phenomena on degree distributions on the Web graph. We now turn to the details of trawling this 'cleaned up' data for communities; our goal is to output a non-overlapping maximal set of cores. The data still has over two million potential fans remaining, with over 60 million links to over 20 million potential centers. Since we still have several million potential

fans, we still cannot afford an enumeration algorithm of the form 'for all subsets of i potential fans, and for all subsets of j potential centers, check if a core is induced'. We therefore resort to a number of additional pruning algorithms that eliminate much of these data. Simultaneously, we retain the property that pruned nodes or links cannot be part of any yet-unidentified core. After reducing our data by another order of magnitude in this fashion, we resort to enumeration at the very end.

A critical requirement of algorithms in this phase is the following. The algorithms must be implementable as a small number of steps, where in each step the data is processed in a stream from disk and then stored back after processing onto disk. Main memory is thus used very carefully. The only other operation we employ is a sorting of the data set — an efficient operation in most platforms.

3.1. Iterative pruning

When looking for (i, j) cores, clearly any potential fan with outdegree smaller than j can be pruned and the associated edges deleted from the graph. Similarly, any potential center with in-degree smaller than i can be pruned and the corresponding edges deleted from the graph. This process can be done iteratively: when a fan gets pruned then some of the centers that it points to may have their in-degrees fall below the threshold i and qualify for pruning as a consequence. Similarly when a center gets pruned, a fan that points to it could have its out-degree fall below its threshold of j and qualify for pruning. While there is an obvious way to do this kind of pruning on small data sets, it fails when the data set becomes large and does not fit into main memory. In our case, the data set is too large to fit into main memory. We represent each URL by a 64-bit hash value. (To avoid collision we require about 60 bits.) Thus, each edge in the Web graph will occupy 128 bits in storing a (source page, destination page) pair. Our machines have 512 MB of main memory. This allows us to represent about 40 million edges in main memory, which is more than an order of magnitude too small if the experiment is scaled to the whole Web.

Because of this, it becomes necessary to design pruning algorithms that efficiently stream the data between secondary and main memory. Luckily, our iterative pruning process is reducible to sorting repeatedly. We sort the edge list by source, then stream through the data eliminating fans that have low outdegree. We then sort the result by destination and eliminate the centers that have low in-degree. We sort by source again, and then again by destination and so on until we reach a point when only few fans/centers are eliminated in each iteration. In fact it is not necessary to repeatedly sort: it suffices to remember and index in memory all the pruned vertices (or more generally, if in each pass we only pruned as many vertices as we could hold in main memory). We would then have two data sets, containing identical data, except that in one case the edges are sorted by source and in the other they are sorted by destination. We alternately scan over these data sets, identifying and pruning pages that do not meet the in-degree or outdegree threshold. We hold and index the set of vertices being pruned in each iteration in memory. This results in a significant improvement in execution time, because there are only two calls to a sorting routine.

The fact that this form of pruning can be reduced to a computation on a data stream and sorting is significant. It would be impossible to do this pruning using a method that required the indexing of edges by source, by destination, or both. Such an index would necessarily have to live on disk and accesses to it could prove to be expensive due to the non-locality of disk access. Designing a method which streams data efficiently through main memory is relatively simple in the case of iterative pruning. It is considerably more challenging when the pruning strategy becomes more sophisticated, as in the case of inclusion–exclusion pruning, the strategy we describe next.

3.2. Inclusion–exclusion pruning

The next pruning strategy — which we call *inclusion–exclusion pruning* — has the following useful property: at every step, we either eliminate a page from contention, or we discover (and output) an (i, j) core. Hence the name inclusion–exclusion: at each step we either 'include' a community or we 'exclude' a page from further contention (as either a fan or as a center), by establishing that it cannot be a part of any core. An important benefit of such pruning is that every step represents useful progress,

either in discovering a community or in pruning the data. Note again that the algorithm must be implementable without relying on holding all of the data in main memory.

The fans (resp. centers) that we choose for inclusion–exclusion pruning are those whose out-degrees (resp. in-degrees) are equal to the threshold j (resp. i). It is relatively easy to check if these nodes are part of any (i, j) core. Consider a fan x with out-degree exactly j, and let $N(x)$ denote the set of centers it points to. Then, x is in an (i, j) core if and only if there are $i - 1$ other fans all pointing to each center in $N(x)$. For small values of i and j and given an index on both fans and centers, we can check this condition quite easily. The computation is simply computing the size of the intersection of j sets and checking if the cardinality is at least i. How can we avoid having two indices in main memory?

First notice that we do not need simultaneous access to both indices. The reason is that we can first eliminate fans with out-degree j and then worry about centers with in-degree i. That is, first, from the edge list sorted by the source ID, we detect all the fans with out-degree j. We output for each such fan the set of j centers adjacent to it. We then use an index on the destination ID to generate the set of fans pointing to each of the j centers and to compute the intersection of these sets.

A somewhat more careful investigation reveals that all of these intersection computations can be batched, and therefore, we do not require even one index. From a linear scan of the edges sorted by source find fans of out-degree exactly j. For as many of the fans as will fit in memory, index edges sourced at the fan by destination IDs. At this stage, we have in memory an index by centers which for each center contains fans adjacent to then which have out-degree exactly j. Clearly this is a much smaller index: it contains only edges that are adjacent to fans of out-degree exactly j.

We will now maintain a set of vertices corresponding to each fan whose edges we have indexed. Recall that each retained fan x is adjacent to exactly j centers. This allows us to consider a 'dual' condition equivalent to the condition above.

Fact 2. Let $\{c_1, c_2, \ldots, c_j\}$ be the centers adjacent to x. Let $N(c_t)$ denote the neighborhood of c_t, the set of fans that point to c_t. Then, x is part of a core if and only if the intersection of the sets $N(c_t)$ has size at least i.

We will use this fact to determine which of the fans that we have chosen qualify as part of a community. If the fan qualifies, then we output the community; otherwise we output nothing. In either case, we can prune the fan. It turns out that the condition in Fact 2 can be verified efficiently by batching many fans together. This is exactly what we do.

We maintain a set $S(x)$ corresponding to each fan x. The goal is that at the end of the computation, the set corresponding to the fan x will be the intersection of the sets $N(c_t)$ specified in Fact 2, above. Stream through the edges sorted by destination. For each destination y check if it is in the small index. If so, then there is at least one fan of outdegree exactly j adjacent to it. If not, edges adjacent to y are meaningless in the current pass. Assume, therefore that y is in the index. For each degree j fan x adjacent to y, intersect the set of fans adjacent to y with the set corresponding to x. That is, $S(x)$ is intersected with $N(y)$. Recall that since the edges are sorted by destination, $N(y)$ is available as a contiguous sequence in the scan.

At the end of this batched run, $S(x)$ is the set required to verify Fact 2 for every x. For vertices x whose sets, $S(x)$, have size at least i, the $S(x)$ corresponds to the fans in the cores they belong in. In this case, we can output the community immediately. In either case, we can prune x. We can also (optionally) prune all the fans that belong in some community that we output. Thus, we get the following interesting fact. Inclusion–exclusion pruning can be reduced to two passes over the data set separated by a sort operation.

The following statements apply to all the pruning steps up until this point.

Theorem 1. *Given any graph, no yet-unidentified cores are eliminated by the above pruning steps.*

Theorem 2. *The running time of the above pruning steps is linear in the size of the input plus the number of communities produced in these steps.*

Theorem 1 states that the set of cores generated so far is *complete* in that no cores are missed. This

is not always a desirable property, especially if many 'false positives' are produced. However, our results in Section 4 show that we do not suffer from this problem. Theorem 2 shows that our algorithms up until this point are *output-sensitive*: the running time (besides being linear in the size of the input) grows linearly in the size of the output. In other words, we spend constant time per input item that we read, and constant time per core we produce (if we view i and j as constants independent of the size of the input).

3.3. Core generation and filtering

The inclusion–exclusion pruning step generates a number of cores. Here is an instance of the set of fans that form a community core. The community in question turns out (upon inspection) to be Muslim student organizations at U.S. universities:

http://chestnut.brown.edu
/Students/Muslim_Students/Org.html
http://wings.buffalo.edu
/sa/muslim/resources/msa.html
http://gopherhost.cc.utexas.edu
/students/msa/links/links.html

Table 1 shows the number of cores that were output during inclusion–exclusion pruning, for various values of i and j. Communities with any fixed value of j are largely disjoint due to the way the inclusion–exclusion pruning is done. Thus, our trawling has extracted about 135 KB communities (summing up communities with $j = 3$).

Table 1
Number of cores

i	j	Nr. of cores	Nr. of non-nepotistic cores
3	3	89565	38887
3	5	70168	30299
3	7	60614	26800
3	9	53567	24595
4	3	29769	11410
4	5	21598	12626
4	7	17754	10703
4	9	15258	9566
5	3	11438	7015
5	5	8062	4927
5	7	6626	4071
5	9	5684	3547
6	3	4854	2757
6	5	3196	1795
6	7	2549	1425
6	9	2141	1206

Next we filtered away *nepotistic cores*. A nepotistic core is one where some of the fans in the core come from the same Website. The underlying principle is that if many of the fans in a core come from the same Website, this may be an artificially established community serving the ends (very likely commercial) of a single entity, rather than a spontaneously emerging Web community. For this purpose, the following definition of 'same Website' is used. If the site contains at most three fields, for instance, yahoo.com, or www.ibm.com then the site is left as is. If the site has more than three fields, as in www3.yahoo.co.uk, then the first field is dropped. The last column in the table above represents the number of non-nepotistic cores. As can be seen the number of nepotistic cores is significant, but not overwhelming — on our 18-month old crawl. About half the cores pass the nepotism test.

We would expect the number of cores in the current Web to be significantly higher.

3.4. Finishing it off

After the inclusion–exclusion pruning step, we are left with about 5 million unpruned edges when we are looking for $(3, 3)$ cores (the case when the largest number of unpruned edges is left). We can now afford to enumerate the remaining cores using existing techniques; we briefly describe this final step now. We build cores iteratively and identify a core by enumerating its fans. Fix a value for j, and note that every proper subset of the fans in any (i, j) core forms a core of smaller size. This fact, together with the diminished size of the data set, allows us to run the *a priori* algorithm of Agrawal and Srikant [2], as follows.

Fix j. Start with all $(1, j)$ cores. This is simply the set of all vertices with outdegree at least j. Now, construct all $(2, j)$ cores by checking every fan which also cites any center in a $(1, j)$ core. Compute all $(3, j)$ cores likewise by checking every fan which cites any center in a $(2, j)$ core, and so on. Fig. 2 plots the number of resulting cores as a function of i and j, based on the subgraph

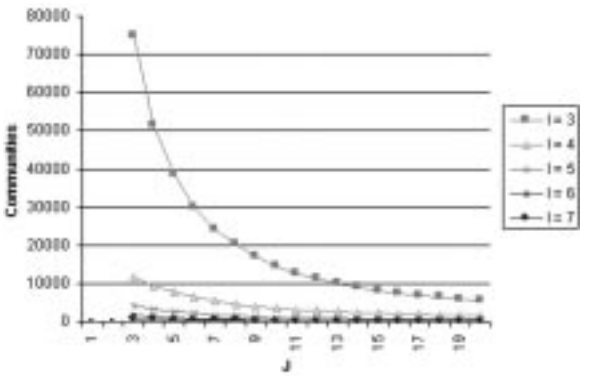

Fig. 2. Number of cores remaining after pruning.

remaining after the (3, 3) pruning step. Note that, in contrast to the pruning algorithm, this algorithm will output cores and all their subcores. We have normalized so that the count of (i, j) cores includes only non-overlapping cores, but the (i, j) cores may overlap with the (i', j') cores. The number of (3, 3) cores is about 75 KB, and is an upper bound on the total number of disjoint cores of size (3, 3) or greater remaining after inclusion–exclusion pruning.

Curiously, the number (75 KB) is smaller than, though comparable to, the number of cores found by the inclusion–exclusion step (about 135 KB). Thus, we note that there are approximately 200 KB potential communities in our data set; in fact, as our results in the next section show, it appears that virtually all of these cores correspond to real communities (rather than coincidental occurrences of complete bipartite subgraphs). Given that the data set is about a year and a half old, it seems safe to conjecture that there are at least as many communities in the Web today.

4. Evaluations of communities

In this section we describe our preliminary evaluations of the communities that have been trawled by our system; as a first step, we rely on manual inspection to study the communities that result. (In the longer run we need a mechanized process for dealing with over a hundred thousand communities. This raises significant challenges in information retrieval research.) For this manual inspection, we have picked a random sample of 400 cores (200 (3, 3) cores and 200 (3, 5) cores) from our list.

Fossilization. Given that we worked with an 18-month old crawl, the first question we studied was how many of the community cores that we had discovered were recoverable in today's Web. A *fossil* is a community core not all of whose fan pages exist on the Web today. To our surprise, many of the community cores were recoverable as communities in today's Web. Of the 400 communities in the random sample, 122, or roughly 30% were fossilized. The rest were still alive. Given prevailing estimates of the half-life of Web pages (well under 6 months), we found it surprising that fan pages corresponding to community cores are significantly longer-lived. This seems to be yet another indicator of the value of resource collections in the Web and consequently, the robustness of methods such as ours.

Communities. The next question was in studying the communities themselves. To give a flavor of communities we identify, we present the following two examples. The first one deals with Japanese pop singer *Hekiru Shiina*. The following are the fans of this community:

http://awa.a-web.co.jp/~buglin/shiina/link.html
http://hawk.ise.chuo-u.ac.jp/student/person/tshiozak/hobby/heki/hekilink.html
http://noah.mtl.t.u-tokyo.ac.jp/~msato/hobby/hekiru.html

The next example deals with Australian fire brigade services. The following are the fans of this community:

http://maya.eagles.bbs.net.au/~mp/aussie.html
http://homepage.midusa.net/~timcorny/intrnatl.html
http://fsinfo.cs.uni-sb.de/~pahu/links_australien.html

A more extensive annotated list of some the communities that we find can be viewed at theory.stanford.edu/~raghavan/comm.html

Reliability. The next question was what fraction of the live cores were still cogent in that they covered a single characterizable theme. We found the cores to be surprisingly reliable. Of the 400 cores, there were 16 coincidental cores — a collection of fan pages without any cogent theme unifying them. This amounts to just 4% of our trawled cores. Given the scale of

Table 2
A community of Australian fire brigades

Authorities	Hubs
NSW Rural Fire Service Internet Site	New South Wales Fir...ial Australian Links
NSW Fire Brigades	Feuerwehrlinks Australien
Sutherland Rural Fire Service	FireNet Information Network
CFA: County Fire Authority	The Cherrybrook Rur...re Brigade Home Page
"The National Cente...ted Children's Ho...	New South Wales Fir...ial Australian Links
CRAFTI Internet Connexions-INFO	Fire Departments, F... Information Network
Welcome to Blackwoo... Fire Safety Serv...	The Australian Firefighter Page
The World Famous Guestbook Server	Kristiansand brannv...dens brannvesener...
Wilberforce County Fire Brigade	Australian Fire Services Links
NEW SOUTH WALES FIR...ES 377 STATION	The 911 F,P,M., Fir...mp; Canada A Section
Woronora Bushfire Brigade	Feuerwehrlinks Australien
Mongarlowe Bush Fire – Home Page	Sanctuary Point Rural Fire Brigade
Golden Square Fire Brigade	Fire Trails "l...ghters around the...
FIREBREAK Home Page	FireSafe – Fire and Safety Directory
Guises Creek Volunt...fficial Home Page...	Kristiansand Firede...departments of th...

the Web and our usage of link information alone, one might expect a far larger fraction of accidental communities. It appears that our many careful pruning steps paid off here. Given these sample results, one can extrapolate that the fraction of fossils and coincidences together account for less than 35% of the cores that were trawled. In other words, we estimate that we have extracted some 130 KB communities that are alive and cogent in today's Web.

Recoverability. The core is only part of the community. For communities that have not fossilized, can we recover today's community from the core that we extracted? A method we use for this purpose is to run the Clever search engine on the community core, using the fans as exemplary hubs (and no text query). For details on how this is done, see [9,16]. Table 2 shows the output of Clever (top 15 hubs and authorities) run on one of our cores, which turns out to be for Australian fire brigades. A more extensive list can be found at theory.stanford.edu/~raghavan/brecca.html

Quality. How many of our communities are unknown to explicit ontological efforts? We found that of the sampled communities, 56% were not in Yahoo! as reconstructed from our crawl and 29% are not in Yahoo! today in any form whatsoever. (The remaining are present in Yahoo! today, though in many cases Yahoo! has only one or two URLs in its listing whereas our process frequently yields many more. We interpret this finding as a measure of reliability of the trawling process, namely, that many of the communities that we had found *emerging* 18 months ago have now *emerged*. Also, none of the trawled communities appears in less than the third level of the Yahoo! hierarchy, with the average level (amongst those present in Yahoo!) being about 4.5 and many communities that were as deep as 6 in the current Yahoo! tree. We believe that trawling a current copy of the Web will result in the discovery of many more communities that will become explicit in the future.

Open problems. The dominant open problems are automatically extracting semantic information and organizing the communities we trawl into useful structure. Another interesting problem is to trawl successive copies of the Web, tracking the sociology of emerging communities.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom and J. Weiner, The Lorel Query language for semistructured data, International J. Digital Libraries, 1(1) (1997) 68–88.
[2] R. Agrawal and R. Srikanth, Fast algorithms for mining association rules, in: Proc. VLDB, Santiago, Chile, September 1994.
[3] K. Bharat and A. Broder, A technique for measuring the relative size and overlap of public Web search engines, in: Proc. 7th Int. World Wide Web Conference, Brisbane, Australia, Elsevier, Amsterdam, April 1998, pp. 379–388.
[4] K. Bharat and M. Henzinger, Improved algorithms for topic

distillation in hyperlinked environments, in: Proc. 21st SIGIR Conference, Melbourne, Australia, 1998.
[5] S. Brin and L. Page, The anatomy of a large scale hypertextual web search engine, in: Proc. 7th Int. World Wide Web Conference, Brisbane, Australia, April 1998, pp. 107–117.
[6] A. Broder, S. Glassman, M. Manasse and G. Zweig, Syntactic clustering of the Web, in: Proc. 6th Int. World Wide Web Conference, April 1997, pp. 391–404.
[7] J. Carriere and R. Kazman, WebQuery: searching and visualizing the Web through connectivity, in: Proc. 6th Int. World Wide Web Conference, 1997.
[8] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan and S. Rajagopalan, Automatic resource compilation by analyzing hyperlink structure and associated text, in: Proc. 7th World-Wide Web Conference, 1998, pp. 65–74.
[9] S. Chakrabarti, B. Dom, D. Gibson, S.R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins, Experiments in topic distillation, in: Proc. ACM SIGIR Workshop on Hypertext Information Retrieval on the Web, Melbourne, Australia, 1998.
[10] S. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas and R.A. Harshman, Indexing by latent semantic analysis, Journal of the Society for Information Science, 41(6), 391–407.
[11] D. Florescu, A. Levy, A. Mendelzon, Database techniques for the World-Wide Web: a survey, SIGMOD Rec. 27(3) (1998) 59–74.
[12] D. Gibson, J. Kleinberg, P. Raghavan, Inferring Web communities from link topology, in: Proc. 9th ACM Conf. on Hypertext and Hypermedia, 1998.
[13] M. Henzinger, P. and S. Rajagopalan, Computing on data streams, AMS–DIMACS series, Special Issue on Computing on Very Large Datasets, also Technical Note 1998-011, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, May 1998.
[14] J. Kleinberg, Authoritative sources in a hyperlinked environment, in: Proc. 9th ACM–SIAM Symp. on Discrete Algorithms (SODA), January 1998.
[15] D. Konopnicki and O. Shmueli, Information gathering on the World Wide Web: the W3QL query language and the W3QS system, Transactions on Database Systems, September 1998.
[16] R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins, Human effort in semi-automated taxonomy construction, submitted for publication.
[17] L.V.S. Lakshmanan, F. Sadri and I.N. Subramanian, A declarative approach to querying and restructuring the World-Wide-Web, in: Post-ICDE Workshop on Research Issues in Data Engineering (RIDE'96), New Orleans, February 1996.
[18] A. Mendelzon, G. Mihaila, T. Milo, Querying the World Wide Web, Journal of Digital Libraries 1(1) (1997) 68–88.
[19] P. Pirolli, J. Pitkow and R. Rao, Silk from a sow's ear: extracting usable structures from the Web, in: Proc. ACM SIGCHI Conf. on Human Factors in Computing, 1996.
[20] E. Spertus, ParaSite: mining structural information on the Web, in: Proc. 6th Int. World Wide Web Conference, 1997.
[21] D. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov and A. Rosenthal, Query Flocks: a generalization of association rule mining, in: Proc. 1998 ACM SIGMOD Conf. on Management of Data, 1998.
[22] H.D. White and K.W. McCain, Bibliometrics, in: Annual Review of Information Science and Technology, Elsevier, 1989, pp. 119–186.

Ravi Kumar received his Ph.D. in Computer Science from Cornell University in 1998 and since then he has been a Research Staff Member at the IBM Almaden Research Center. His research interests include randomization, complexity theory, and information processing.

Prabhakar Raghavan received his Ph.D. in Computer Science from the University of California, Berkeley in 1986. Since then he has been a Research Staff Member at IBM Research, first at the Thomas J. Watson Research Center in Yorktown Heights, NY, and since 1995 at the Almaden Research Center in San Jose, California. He is also a Consulting Professor at the Computer Science Department, Stanford University. His research interests include algorithms, randomization, information retrieval and optimization. He is the co-author of a book *Randomized Algorithms* and numerous technical papers. He has served on the editorial boards and program committees of a number of journals and conferences.

Sridhar Rajagopalan has received a B.Tech. from the Indian Institute of Technology, Delhi, in 1989 and a Ph.D. from the University of California, Berkeley in 1994. He spent two years as a DIMACS postdoctoral fellow at Princeton University. He is now a Research Staff Member at the IBM Almaden Research Center. His research interests include algorithms and optimization, randomization, information and coding theory and information retrieval.

Andrew Tomkins received his BSc in mathematics and computer science from MIT in 1989, and his Ph.D. from CMU in 1997. He now works at the IBM Almaden Research Center in the Principles and Methodologies group. His research interests include algorithms, particularly online algorithms, disk scheduling and prefetching, pen computing and OCR, and the World Wide Web.